XML-Documents and Tabments

1 An Introducing Example

The following XML document, pupils.xml, includes nine pupils:

```
<!DOCTYPE PUPILS [
<!ELEMENT PUPILS (PUPIL*)>
<!ELEMENT PUPIL (NAME, FIRSTNAME, SUBJECTTUP*)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT FIRSTNAME (#PCDATA)>
<!ELEMENT SUBJECTTUP (SUBJECT, MARK*)>
<!ELEMENT SUBJECT (#PCDATA)>
<!ELEMENT MARK (#PCDATA)>
]>
    <PUPILS>
    <PUPIL>
        <NAME>Meier</NAME>
        <FIRSTNAME>Hans Peter Paul</FIRSTNAME>
        <SUBJECTTUP>
                <SUBJECT>Maths</SUBJECT>
                <MARK>1</MARK>
                <MARK>1</MARK>
                <MARK>1</MARK>
        </SUBJECTTUP>
    </PUPIL>
    <PUPIL>
        <NAME>Mueller</NAME>
        <FIRSTNAME>Antje</FIRSTNAME>
        <SUBJECTTUP>
                <SUBJECT>Maths</SUBJECT>
                <MARK>2</MARK>
                <MARK>1</MARK>
        </SUBJECTTUP>
        <SUBJECTTUP>
                <SUBJECT>German</SUBJECT>
                <MARK>4</MARK>
                <MARK>1</MARK>
                <MARK>2</MARK>
                <MARK>4</MARK>
                <MARK>1</MARK>
                <MARK>2</MARK>
        </SUBJECTTUP>
    </PUPIL>
    <PUPIL>
        <NAME>Schulz</NAME>
        <FIRSTNAME>Michael</FIRSTNAME>
        <SUBJECTTUP>
                <SUBJECT>Chemistry</SUBJECT>
                <MARK>1</MARK>
                <MARK>4</MARK>
```

```
            <MARK>4</MARK>
            <MARK>4</MARK>
            <MARK>4</MARK>
            <MARK>4</MARK>
            <MARK>4</MARK>
            <MARK>4</MARK>
            <MARK>4</MARK>
            <MARK>4</MARK>
        </SUBJECTTUP>
        <SUBJECTTUP>
            <SUBJECT>German</SUBJECT>
            <MARK>1</MARK>
            <MARK>4</MARK>
            </SUBJECTTUP>
    </PUPIL>
    <PUPIL>
        <NAME>Schulz</NAME>
        <FIRSTNAME>Michael</FIRSTNAME>
        <SUBJECTTUP>
            <SUBJECT>German</SUBJECT>
            <MARK>1</MARK>
            <MARK>1</MARK>
            <MARK>1</MARK>
            <MARK>2</MARK>
            <MARK>2</MARK>
            <MARK>1</MARK>
            <MARK>1</MARK>
            <MARK>1</MARK>
            <MARK>1</MARK>
        </SUBJECTTUP>
        <SUBJECTTUP>
            <SUBJECT>Maths</SUBJECT>
            <MARK>1</MARK>
            <MARK>4</MARK>
        </SUBJECTTUP>
    </PUPIL>
    <PUPIL>
        <NAME>Schmidt</NAME>
        <FIRSTNAME>Max</FIRSTNAME>
        <SUBJECTTUP>
            <SUBJECT>Maths</SUBJECT>
            <MARK>1</MARK>
            <MARK>4</MARK>
            <MARK>2</MARK>
            <MARK>1</MARK>
            <MARK>4</MARK>
            <MARK>2</MARK>
            <MARK>1</MARK>
            <MARK>4</MARK>
            <MARK>2</MARK>
        </SUBJECTTUP>
```

```
    <SUBJECTTUP>
            <SUBJECT>Chemistry</SUBJECT>
            <MARK>4</MARK>
    </SUBJECTTUP>
</PUPIL>
<PUPIL>
    <NAME>Mayer</NAME>
    <FIRSTNAME>Fritz</FIRSTNAME>
    <SUBJECTTUP>
            <SUBJECT>Maths</SUBJECT>
            <MARK>3</MARK>
            <MARK>1</MARK>
            <MARK>3</MARK>
            <MARK>1</MARK>
            <MARK>3</MARK>
            <MARK>1</MARK>
            <MARK>3</MARK>
            <MARK>1</MARK>
            <MARK>3</MARK>
            <MARK>1</MARK>
    </SUBJECTTUP>
    <SUBJECTTUP>
            <SUBJECT>Physics</SUBJECT>
            <MARK>2</MARK>
    </SUBJECTTUP>
</PUPIL>
<PUPIL>
    <NAME>Chang</NAME>
    <FIRSTNAME>Lee</FIRSTNAME>
    <SUBJECTTUP>
            <SUBJECT>Maths</SUBJECT>
            <MARK>2</MARK>
            <MARK>2</MARK>
            <MARK>1</MARK>
    </SUBJECTTUP>
</PUPIL>
<PUPIL>
    <NAME>Perner</NAME>
    <FIRSTNAME>Nadin</FIRSTNAME>
    <SUBJECTTUP>
            <SUBJECT>German</SUBJECT>
            <MARK>1</MARK>
    </SUBJECTTUP>
    <SUBJECTTUP>
            <SUBJECT>Maths</SUBJECT>
    </SUBJECTTUP>
</PUPIL>
<PUPIL>
    <NAME>Mayer</NAME>
    <FIRSTNAME>Fritz</FIRSTNAME>
</PUPIL>
```

```
</PUPILS>
```

The third pupil, Michael Schulz, has 10 respectively 2 marks in two specialized subjects (Chemistry and German). Each pupil having name, first name and specialized subjects is framed by a PUPIL-marking (tag). Each subject with the appropriate marks is framed by one SUBJECTTUP tag. "TUP" comes from TUPLE. The XML-representation of data is **conceptually** very favourable, since one recognizes exactly that a number is a MARK and which word is a name, first name or subject. However, the above XML document needs a lot of space, which can be saved in the following way in form of a tabment table. Thus, this document fits comfortably on a (small) screen or a small sheet paper.

```
<<  L( NAME,      FIRSTNAME, L(   SUBJECT,     L(  MARK))):: 
       Meier      Hans Peter Paul Maths          1 1 1
       Mueller    Antje           Maths          2 1
                                  German         4 1 2 4 1 2
       Schulz     Michael         Chemistry      1 4 4 4 4 4 4 4 4 4
                                  German         1 4
       Schulz     Michael         German         1 1 1 2 2 1 1 1 1
                                  Maths          1 4
       Schmidt    Max             Maths          1 4 2 1 4 2 1 4 2
                                  Chemistry      4
       Mayer      Fritz           Maths          3 1 3 1 3 1 3 1 3 1
                                  Physics        2
       Chang      Lee             Maths          2 2 1
       Perner     Nadin           German         1
                                  Maths
       Mayer      Fritz>>
```

Here, the range of one column covers the first letter of the column header ("F" of first name) to one letter before the next column. In the first line the first name "Hans Peter Paul" belongs completely to the first name column. This tabment is fully equivalent to the above XML document, if it is clear that it has the same meta data. In our case the meta data (otto-DTD) looks as follows:

```
<<  M( TAG,          TYPE)::
       TABMENT       PUPILS
       PUPILS        L(PUPIL)
       PUPIL         NAME, FIRSTNAME, L(SUBJECTTUP)
       SUBJECTTUP    SUBJECT, L(MARK)
       SUBJECT       TEXT
       NAME          TEXT
       FIRSTNAME     TEXT
       MARK          ZAHL>>
```

The Otto-DTD (DTD=document type definition) expresses that the tabment is enclosed by one PUPILS-tag, that PUPILS is a list (L) of PUPIL data, that a PUPIL is a triple from NAME, FIRSTNAME and a list (L) of SUBJECTTUPles, and a SUBJECTTUPle is a pair of a SUBJECT and a list of MARKs. NAME, FIRSTNAME and SUBJECT are atomic text data and MARK is an integer (= "Zahl" in German). Because of reasons of uniformity we present the meta data as tabments again. If these meta data belong to the above tabment, then the computer-internal representation of the tabments also includes the tags PUPILS, PUPIL, and SUBJECTTUP, although these are not visible. The use of these tags in the tabular

representation would clearly injure the representation of the table. We see that an otto-DTD is a little simpler than an XML-DTD. We use an XML-attribut in the same way like an XML-element; we only mark it at the beginning by "@".

M: set (German: Menge)

L: list

B: bag (multi-set)

It is also possible to represent the above pupil tabment by a simpler flat table:

<<L(NAME,	FIRSTNAME,	SUBJECT,	MARK)::
Meier	Hans Peter Paul	Mathe	1
Meier	Hans Peter Paul	Mathe	1
Meier	Hans Peter Paul	Mathe	1
Mueller	Antje	Mathe	2
Mueller	Antje	Mathe	1
Mueller	Antje	Deutsch	4
Mueller	Antje	Deutsch	1
Mueller	Antje	Deutsch	2
Mueller	Antje	Deutsch	4
Mueller	Antje	Deutsch	1
Mueller	Antje	Deutsch	2
Schulz	Michael	Chemie	1
Schulz	Michael	Chemie	4
Schulz	Michael	Chemie	4
Schulz	Michael	Chemie	4
Schulz	Michael	Chemie	4
Schulz	Michael	Chemie	4
Schulz	Michael	Chemie	4
Schulz	Michael	Chemie	4
Schulz	Michael	Chemie	4
Schulz	Michael	Chemie	4
Schulz	Michael	Deutsch	1
Schulz	Michael	Deutsch	4
Schulz	Michael	Deutsch	1
Schulz	Michael	Deutsch	1
Schulz	Michael	Deutsch	1
Schulz	Michael	Deutsch	2
Schulz	Michael	Deutsch	2
Schulz	Michael	Deutsch	1
Schulz	Michael	Deutsch	1
Schulz	Michael	Deutsch	1
Schulz	Michael	Deutsch	1
Schulz	Michael	Mathe	1
Schulz	Michael	Mathe	4
Schmidt	Max	Mathe	1
Schmidt	Max	Mathe	4
Schmidt	Max	Mathe	2
Schmidt	Max	Mathe	1
Schmidt	Max	Mathe	4
Schmidt	Max	Mathe	2
Schmidt	Max	Mathe	1
Schmidt	Max	Mathe	4
Schmidt	Max	Mathe	2

Schmidt	Max	Chemie	4
Mayer	Fritz	Mathe	3
Mayer	Fritz	Mathe	1
Mayer	Fritz	Mathe	3
Mayer	Fritz	Mathe	1
Mayer	Fritz	Mathe	3
Mayer	Fritz	Mathe	1
Mayer	Fritz	Mathe	3
Mayer	Fritz	Mathe	1
Mayer	Fritz	Mathe	3
Mayer	Fritz	Mathe	1
Mayer	Fritz	Physik	2
Chang	Lee	Mathe	2
Chang	Lee	Mathe	2
Chang	Lee	Mathe	1
Perner	Nadin	Deutsch	1>>

A further, more convenient “XML” representation seems to be the following one:

```
<PUPILS>
 <PUPIL>
  <NAME>Meier</NAME>
  <FIRSTNAME>Hans Peter Paul</FIRSTNAME>
  <SUBJECTTUP>
   <SUBJECT>Maths</SUBJECT>
   <MARKS>1 1 1</MARKS>
  </SUBJECTTUP>
 </PUPIL>
 <PUPIL>
  <NAME>Mueller</NAME>
  <FIRSTNAME>Antje</FIRSTNAME>
  <SUBJECTTUP>
   <SUBJECT>Maths</SUBJECT>
   <L(MARK)>2 1</L(MARK)>
  </SUBJECTTUP>
  <SUBJECTTUP>
   <SUBJECT>German</SUBJECT>
   <L(MARK)>4 1 2 4 1 2</L(MARK)>
  </SUBJECTTUP>
 </PUPIL>
 <PUPIL>
  <NAME>Schulz</NAME>
  <FIRSTNAME>Michael</FIRSTNAME>
  <SUBJECTTUP>
   <SUBJECT>Chemistry</SUBJECT>
   <L(MARK)>1 4 4 4 4 4 4 4 4 4</L(MARK)>
  </SUBJECTTUP>
  <SUBJECTTUP>
   <SUBJECT>German</SUBJECT>
   <L(MARK)>1 4</L(MARK)>
  </SUBJECTTUP>
```

```
</PUPIL>
<PUPIL>
 <NAME>Schulz</NAME>
 <FIRSTNAME>Michael</FIRSTNAME>
 <SUBJECTTUP>
  <SUBJECT>German</SUBJECT>
   <L(MARK)>1 1 1 2 2 1 1 1 1</L(MARK)>
 </SUBJECTTUP>
 <SUBJECTTUP>
   <SUBJECT>Maths</SUBJECT>
   <L(MARK)>1 4</L(MARK)>
 </SUBJECTTUP>
</PUPIL>
<PUPIL>
 <NAME>Schmidt</NAME>
 <FIRSTNAME>Max</FIRSTNAME>
 <SUBJECTTUP>
   <SUBJECT>Maths</SUBJECT>
   <L(MARK)>1 4 2 1 4 2 1 4 2</L(MARK)>
 </SUBJECTTUP>
 <SUBJECTTUP>
  <SUBJECT>Chemistry</SUBJECT>
   <L(MARK)>4</L(MARK)>
 </SUBJECTTUP>
</PUPIL>
<PUPIL>
 <NAME>Mayer</NAME>
 <FIRSTNAME>Fritz</FIRSTNAME>
 <SUBJECTTUP>
   <SUBJECT>Maths</SUBJECT>
   <L(MARK)>3 1 3 1 3 1 3 1 3 1</L(MARK)>
 </SUBJECTTUP>
 <SUBJECTTUP>
  <SUBJECT>Physics</SUBJECT>
   <L(MARK)>2</L(MARK)>
 </SUBJECTTUP>
</PUPIL>
<PUPIL>
 <NAME>Chang</NAME>
 <FIRSTNAME>Lee</FIRSTNAME>
 <SUBJECTTUP>
   <SUBJECT>Maths</SUBJECT>
   <L(MARK)>2 2 1</L(MARK)>
 </SUBJECTTUP>
</PUPIL>
<PUPIL>
 <NAME>Perner</NAME>
 <FIRSTNAME>Nadin</FIRSTNAME>
 <SUBJECTTUP>
  <SUBJECT>German</SUBJECT>
   <L(MARK)>1</L(MARK)>
```

```
   </SUBJECTTUP>
   <SUBJECTTUP>
     <SUBJECT>Maths</SUBJECT>
   </SUBJECTTUP>
  </PUPIL>
  <PUPIL>
   <NAME>Mayer</NAME>
   <FIRSTNAME>Fritz</FIRSTNAME>
  </PUPIL>
</PUPILS>
```

We see that in this „relational“ representation is also a lot of space wasted (We need 59 rows.) Further names and subjects appear several times (redundancy) and names or subjects with empty sets of subjects resp. marks disappear. To omit this lack we have to introduce null-values. Further, we see that for example the first 3 tuples are equal. This is not allowed in relational data model. Therefore we have to introduce an additional sequence number for each mark. The last improved XML representation requires 93 lines compared with 139 lines of the original XML document.

2 Specification of Tabments in OCAML

We introduce 4 proper collection symbols and one symbol for optional values, which will be treated as sets with at most one element.

```
type coll_sym = Set | Bag | List | S1 | Set_minus | Bag_minus | List_minus;;

type name = string;;                                        (*column names *)

type big_int = Big_int.big_int

type value =
       Bar                                  (* a dash; only in school of interest *)
     | Int_v      of big_int                      (* each big integer is a value *)
     | Float_v    of float
     | Bool_v     of bool
     | String_v   of string;;

type scheme =
       Empty_s                                                  (* empty scheme *)
     | Any                                 (* scheme for an arbitrary tabment *)
     | Any_hierar                            (* all fields of hierarchical path *)
     | Any_flat          (* scheme for an arbitrary tabment, without Coll_t *)
     | Inj            of name                          (* each name is a scheme *)
     | Coll_s         of coll_sym * scheme          (* schemes for collections *)
     | Tuple_s        of scheme list                      (* schemes for tuples *)
     | Alternate_s of scheme list;;                     (* schemes for choice *)
```

Examples:
Coll_s(Set,Tuple_s[Inj"A";Alternate_s[Inj"B1"; Inj"C"]]) corresponds to: M(A, (B1 | C))
Coll_s(S1, Inj "A") corresponds to: A?

```
let type_v = function
```

```
    Bar          -> "BAR"
  | Int_v _      -> ”ZAHL“
  | Float_v _    -> “PZAHL“
  | Bool_v _     -> "BOOL"
  | String_v _ -> “TEXT“;;

type tabment =
    Empty_t                                              (* empty tabment: error value *)
  | El_tab      of value                                 (* an elementary value is a tabment *)
  | Tuple_t     of tabment list                          (* tuple of tabments *)
  | Coll_t      of (coll_sym * scheme * (tabment list) ) (* collection of tabments *)
  | Tag0        of name * tabment                        (* a tabment is enclosed by a name *)
  | Alternate_t of (scheme list) * tabment;;             (* the type of the tabment is
                                                           changed to a choice type *)
```

Coll_t(Set, s,[t1;t2;...;tn]) is a set of elements t1, t2,.., tn, where the types of t1,t2,…,tn have to be equal to the given scheme s. For bags and lists we require also that all elements are of the same type. Alternate_t([s1;s2;s3], t) is of type s1 | s2 | s3 (= Alternate_s [s1;s2;s3]), where t is intended to be of type s1, s2 or s3.

Examples:

t1 = El_tab(String_v “Ernst“) = <<TEXT::Ernst>>
= <TEXT>Ernst</TEXT> = <root>Ernst</root>
t2 = Tag0("NAME", (El_tab(String_v “Clara“))) = <<NAME::Clara>>
= <NAME>Clara</NAME>
t3 = Tag0("NAME", (El_tab(String_v “Josephine“))) = <<NAME::Josephine>>
t4 = Tuple_t [t2; t3] = << NAME, NAME ::
Clara Josephine >>
= <NAME>Clara</NAME>
<NAME>Josephine</NAME>

t5 = Coll_t (List, Inj“NAME“, [t2; t3]) = <<L(NAME) ::
Clara
Josephine>>
= <<L(NAME) ::
Clara Josephine>>
= <NAME>Clara</NAME>
<NAME>Josephine</NAME>
t6 = Tuple_t[Coll_t((S1,Inj “A1”),[Tag0(“A1”,El_tab(Int_v 10))]);Coll_t((S1,Inj “B1”),[])] =
= <<A1?, B1? ::
10 >>
= <A>10</A>
type_t (t1) = Inj “ZAHL” = ZAHL
type_t (t2) = type_t(t3) = Inj “NAME” = NAME
type_t (t4) = Tuple_s[(Inj “NAME”); (Inj “NAME”)] = (NAME, NAME)
type_t (t5) = L(NAME)
type_t (t6) = Tuple_s[Coll_s(S1,Inj”A1”),Coll_s(S1,Inj”B1”)] = (A1?, B1?)

Examples:

dtd_t (t1) = [“TABMENT“, Inj“TEXT“]
dtd_t (t2) = [“TABMENT“, Inj “NAME“; “NAME“, Inj “TEXT“]
dtd_t (t3) = dtd_t (t2)

dtd_t (t4) = [”TABMENT“,Tuple_s[(Inj “NAME”); (Inj “NAME”)]; “NAME“, Inj ”TEXT“]
dtd_t (t5) = [”TABMENT“, Coll_s(List, Inj”NAME”); “NAME”,Inj”TEXT”]
dtd_t (t6) = [”TABMENT“,Tuple_s[Coll_s(S1,Inj”A1”),Coll_s(S1,Inj”B1”)];
“A1”,Inj “ZAHL”]

3 Direct Specification of XML in OCAML

The specification of an XML-DTD is straight forward and lengthy. Therefore we omit it. The specification of an XML-document does not directly require the DTD.

```
type xml =
   | Element of (string * (string * string) list * xml list)
   | PCData of string;;                                  (* each string is an XML-Document *)
```

Element(str, strlist, xmllist) is an XML-document with the root name *str*, where *str* has the attributes of strlist and the children *xmllist*. A pair *(att,v)* of *strlist* is an attribute with name *att* and value *v*.

Examples:

t1 = PCData “15”
t2 = Element(“NAME”,[], [PCData “Paul”])
t3 = Element(“NAME”,[], [PCData “Paula”])
t4 = Element(“root” ,[], [t2; t3]) (* root is an artificially introduced root name *)
t5 = t4
t6 = Element(“A1”, [], [PCData “10”])

Differences between (direct) XML-documents and specified tabments:

1. a) Contrary to XML a tabment has **not** in any case **a root**.
   b) A collection of elements is **not** required to have **a collection tag**.
   c) A tuple of components is **not** required to have **a tuple tag**.
   Therefore database relations can be directly considered as tabments.
   M(A1, A2,..., An) with atomic Ai (i=1,2,...n) is a flat relation scheme.
   M(A1, A2, M(A3, A4, M(A5), M(A6))) with atomic Ai is an example of a nested (NF^2-) relation. Sophisticated selections and other operations can be expressed over tabments, without having these tags.
2. The tabment specification **does not distinguish between attributes and XML-elements**; an attribute is a special element. This is no problem, because attributes can be “tagged” like in XQuery by a preceding “@” Example:
   <a a1=”w1”><a2>w2</a2><a3>w3</a3> =
   Tag0(“a”, Tuple_t[t1; t2; t3]) with t1=Tag0(“@a1”, El_tab(String_v(“w1”))),
   t2=Tag0(“a2”, El_tab(String_v(“w2”))), and t3=Tag0(“a3”,El_tab(String_v(“w3”))
3. In the tabment specification a **tuple** of several elements **is distinguished from a collection** of these elements. This seems to be an advantage, for the specification and implementation of powerful tabment operations (restructuring *stroke*, selection extension, vertical,...). Further we can define the nth component of a tuple. For example, the second component of t6 is empty. If we consider t6 as an XML-document a second component is hard to recognize. In the same way an element name X of type (A1,L(B1)) has two components, but a corresponding X-node may have 1, 2, 3, or more children. If X is of collection type (for example L(A1, B1)) and a corresponding subtabment has n elements (in set theoretic sense), then the corresponding X-node has for example 2*n children. The distinction between tuples

and collections is useful especially for a tabular view, where the components of a tuple are arranged in general horizontally and the elements of a collection vertically. Therefore this distinction is a presupposition for the tabular representation of XML-documents.

4. A simple “untagged” atomic value is also a tabment (El_tab) (and also a XML-document. But a list of simple values like integers is a tabment, they are implicitly tagged by “ZAHL”, but this list is not an XML-document.
5. The specification of tabments handles **additional basic collection types** (Set and Bag).
6. Contrary to XQuery in the tabment-specification **we distinguish between a singleton and the element, which the singleton contains**.
7. Empty collections are visible in the tabment representation, but not in XML.
8. From the specification of tabments a new tabular representation of “XML-documents” is derivable, where not only names but also schemes as tags can be used. This representation may contain hidden tags. Therefore tabular data can be represented often very lucid and comprehensive.
9. The tabment definition is based on **abstract operations** (Tag0, Tuple_t, Coll_t, ...), whereas most scientists feel that XML is based mainly on a **concrete concatenation** operation.